DCache Monitoring

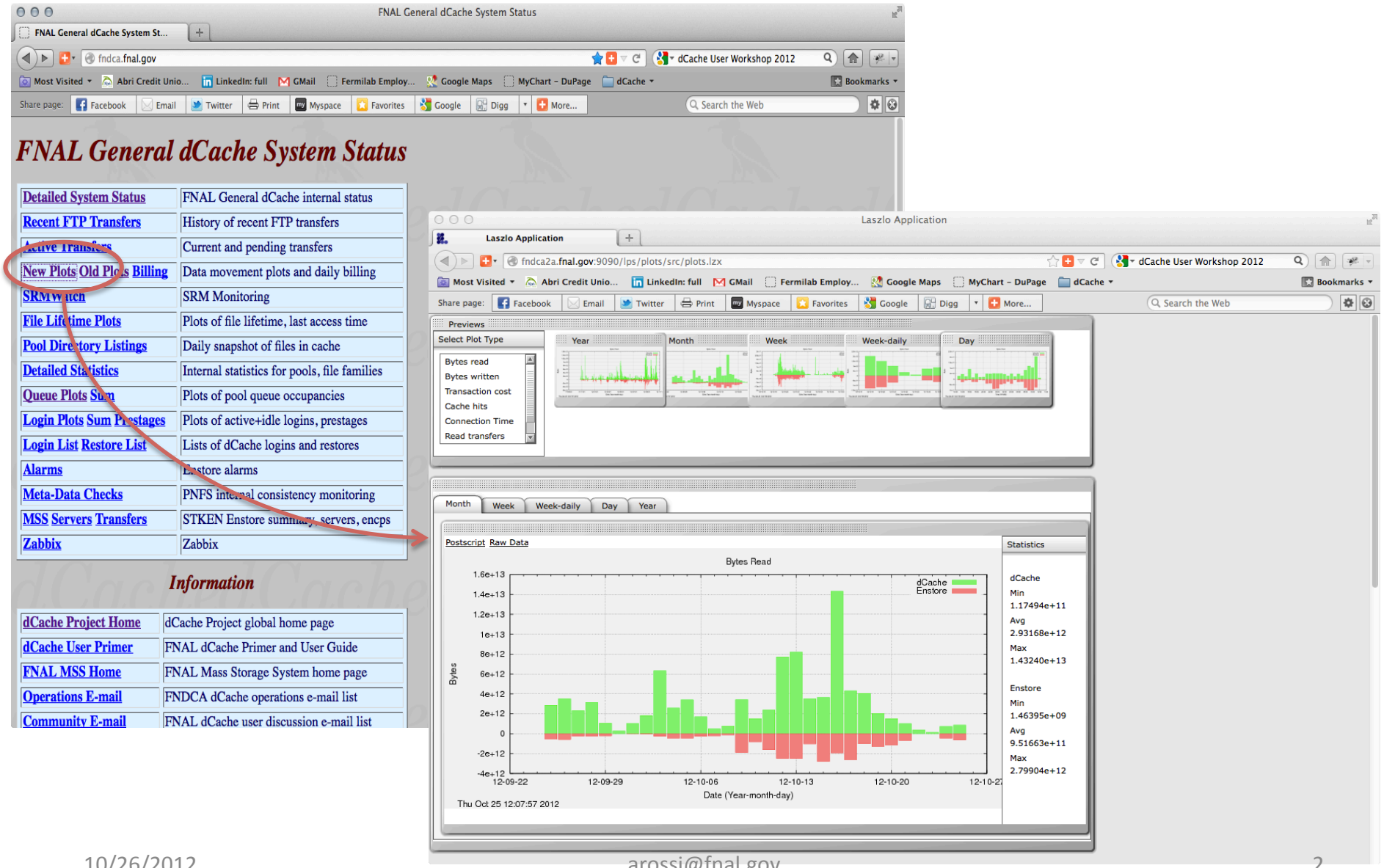
Goal: integrate Fermi monitoring capabilities into dCache itself, in order to:

- reduce external dependencies
- simplify installation, configuration, maintenance
- enhance user experience

Components:

1. Billing histograms
2. Alarms
3. Pool queue load histograms (current project)

Billing Plots: Fermi Public DCache (fndca.fnal.gov)



Billing Histograms

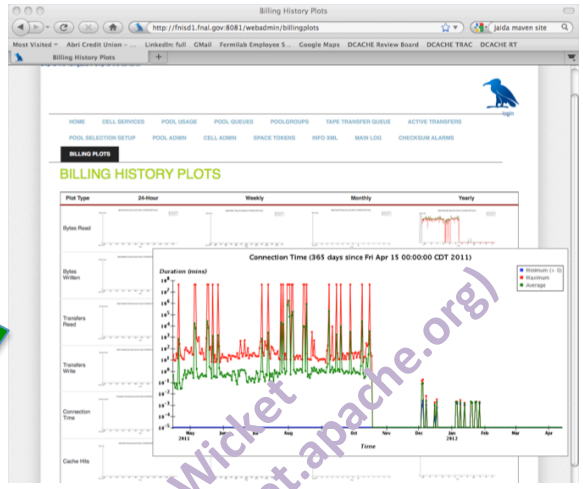
Provide aggregate views of the data for 24-hour, 7-day, 30-day and 365-day periods.

The plot types are:

1. (Giga)bytes read and written for both dCache and HSM backend
2. Number of transactions/transfers for both dCache and HSM backend
3. Maximum, minimum and average connection time
4. Cache hits and misses (optional)

Plots displayed on static web page in dCache 2.2, but on newer “webadmin” servlet page for dCache 2.4+

Built-in DCache Monitoring: Billing Plots



Pool ...

Door ...

cell message

Billing Cell

put

Data Access Object Layer

get

Histogram Generator

write/read

write

Postgre SQL

FS

read

HTTP Server (2288)

Web Admin Servlet

Billing Plots Page

DataNucleus (http://www.datanucleus.org)

JAIDA (http://java.freehep.org)

Wicket (http://wicket.apache.org)

Billing Plots Configuration

Example layout stanza

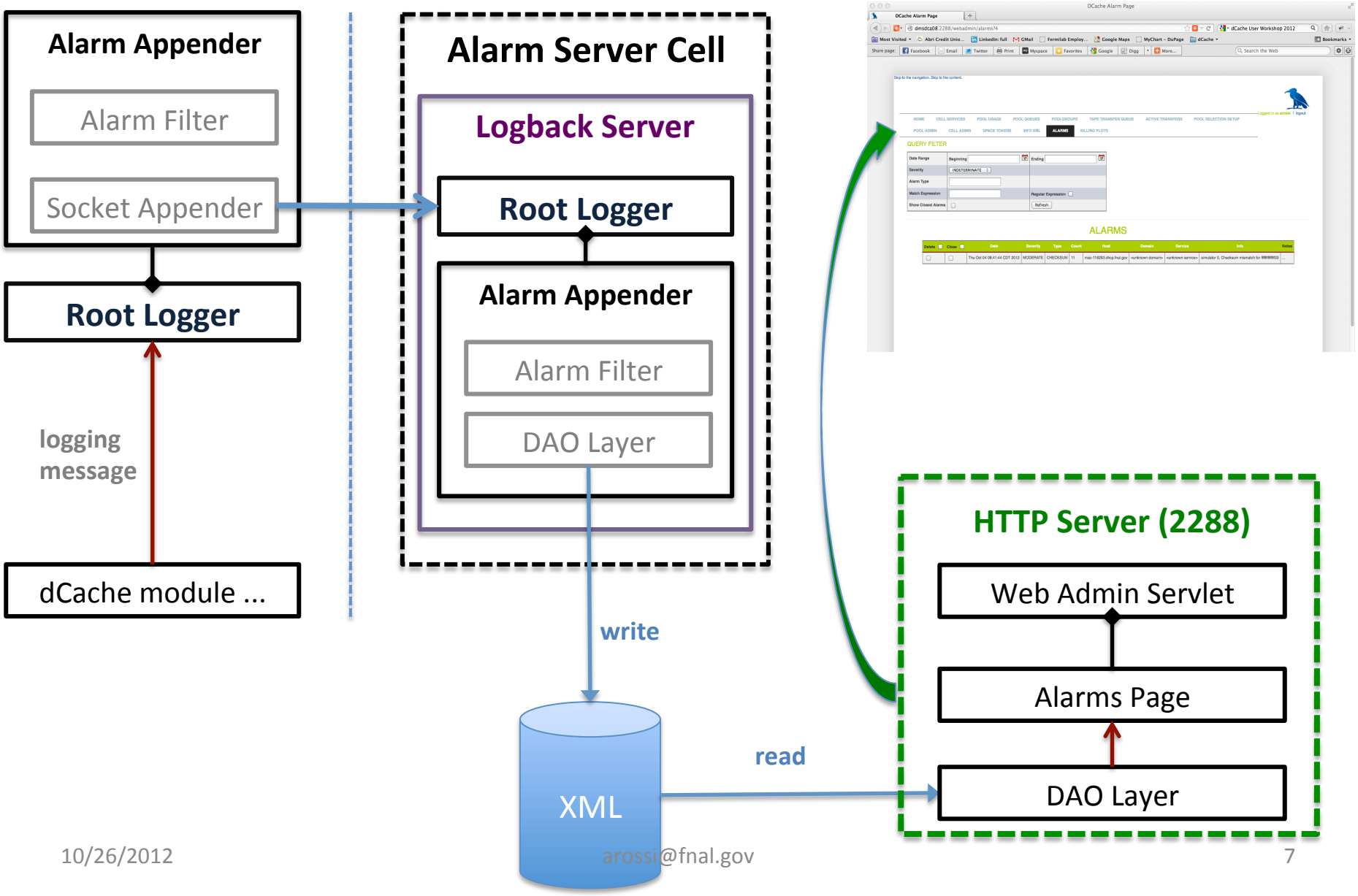
```
## billing plots are turned on
[httpdDomain]
  billingToDb=yes
  generatePlots=true
[httpdDomain/billing]
  billingDbHost=localhost
  billingDbUser=srmdcache
  billingDbPass=
[httpdDomain/httpd]
```

- Plotting depends on presence of database; can be turned off
- Out-of-box configuration should work for most cases
- Plot style can be customized (but this may require some assistance from us)
- The billing service (which generates the plots) must write to a file system that is shared by the domain running the http server
- *For full info, see (forthcoming version of) dCache book, chapter 14*

DCache Alarms

- “Internal” – based on logging messages
- Idea inspired by enstore alarms
- Extensible: not just pre-defined, but alarm types can be easily added by admin (via *logback.xml*)
- Tracked via servlet (“webadmin”) page
- Written to a lightweight (XML) store; automatic cleanup of closed alarms can be enabled/disabled
- Option to configure SMTP appender to send mail notifications
- Configured out-of-box for checksum error alarms (we may add a few more types in future)

Built-in DCache Monitoring: Alarms



Alarms Configuration

Example layout stanzas

```
## should be run in
## its own domain

[alarmserverDomain]
[alarmserverDomain/alarms]

## for webadmin page
[httpdDomain]
  authenticated=true
  webadminAdminGid=1530
[httpdDomain/httpd]
```

- Requires authenticated web page
- *logback.xml* remote appender URL needs to point to host where alarm server domain runs
- other configurable options, but the rest should run out of the box for most cases
- *For full info, see (forthcoming version of) dCache book*
- Using the Alarms web page: **https://dmsdca08.fnal.gov:2288/webadmin/alarms**

New Alarm Definition Example (1)

Suppose you notice repeated errors in a log file, which seem to be symptomatic of a condition which requires rectification; it would be good to get an alarm notification when these occur. To instrument the new alarm,

1. Add the alarm definiton to */etc/dcache/logback.xml*;
2. Restart the domain(s) where the error originates.
3. If you want to receive email for alarms, you also need to configure */var/lib/dcache/alarms/logback-server.xml* and restart the alarmserver domain.

Alarms can be uniquely defined using these categories/properties:

- **level, e.g., ERROR**
- **logger – as defined internally, e.g., `diskCacheV111.namespace.PnfsManagerV3`**
- **regex – on the log message content**
- **regex-flags – for controlling, e.g., case-insensitive, etc.**
- **type – user defined**
- **severity – user defined (one of INDETERMINATE (default), LOW, MODERATE, HIGH, CRITICAL)**
- **thread – also defined internally**
- **include-in-key (defines uniqueness, to avoid duplication): any combination of** *timestamp, message, logger, type, domain, service, host, thread*

In such inductive cases, it is the regex which is going to be the most useful.

New Alarm Definition Example (2)

dcap02-dmsdca07Domain.log:24 Oct 2012 15:24:43 (DCap02-dmsdca07-<unknown>-103) [PnfsManager PnfsGetFileAttributes] fileAttributesNotAvailable: 1 0 client failed 10011 "CacheException(rc=10011;msg=ERROR: function path2inodes(character varying, character varying) does not exist; Hint: No function matches the given name and argument types. You might need to add explicit type casts.; Position: 77)"

Here is how we would add a definition for "missing stored procedure":

```
<appender name="alarms" class="org.dcache.alarms.logback.AlarmDefinitionAppender">
    <!-- this filter determines which events are to be interpreted as alarms;
       the appender converts these into alarm events and passes them
       to its embedded child appender(s)
    -->
    <filter class="org.dcache.alarms.logback.AlarmDefinitionFilter">
        <alarmType>
           logger:org.dcache.pool.classic.ChecksumScanner,
           regex:"Checksum mismatch",
           type:CHECKSUM,
           level:ERROR,
           severity:MODERATE,
           include-in-key:message type host service domain
        </alarmType>
        <alarmType>
           regex:"No function matches the given name and argument types",
           type:MISSING_STORED_PROCEDURE,
           level:ERROR,
           severity:CRITICAL,
           include-in-key:message host
         </alarmType>
    </filter>
    <appender-ref ref="remote"/>
 </appender>
```

Checksum Alarm

New Alarm

Probable identifying string

Full message and database installation identifies type; repeated instances are considered duplicates (no timestamp)

gPlazma

Stands for "**G**rid **Pl**uggable **A**uthori**z**ation **M**odule".

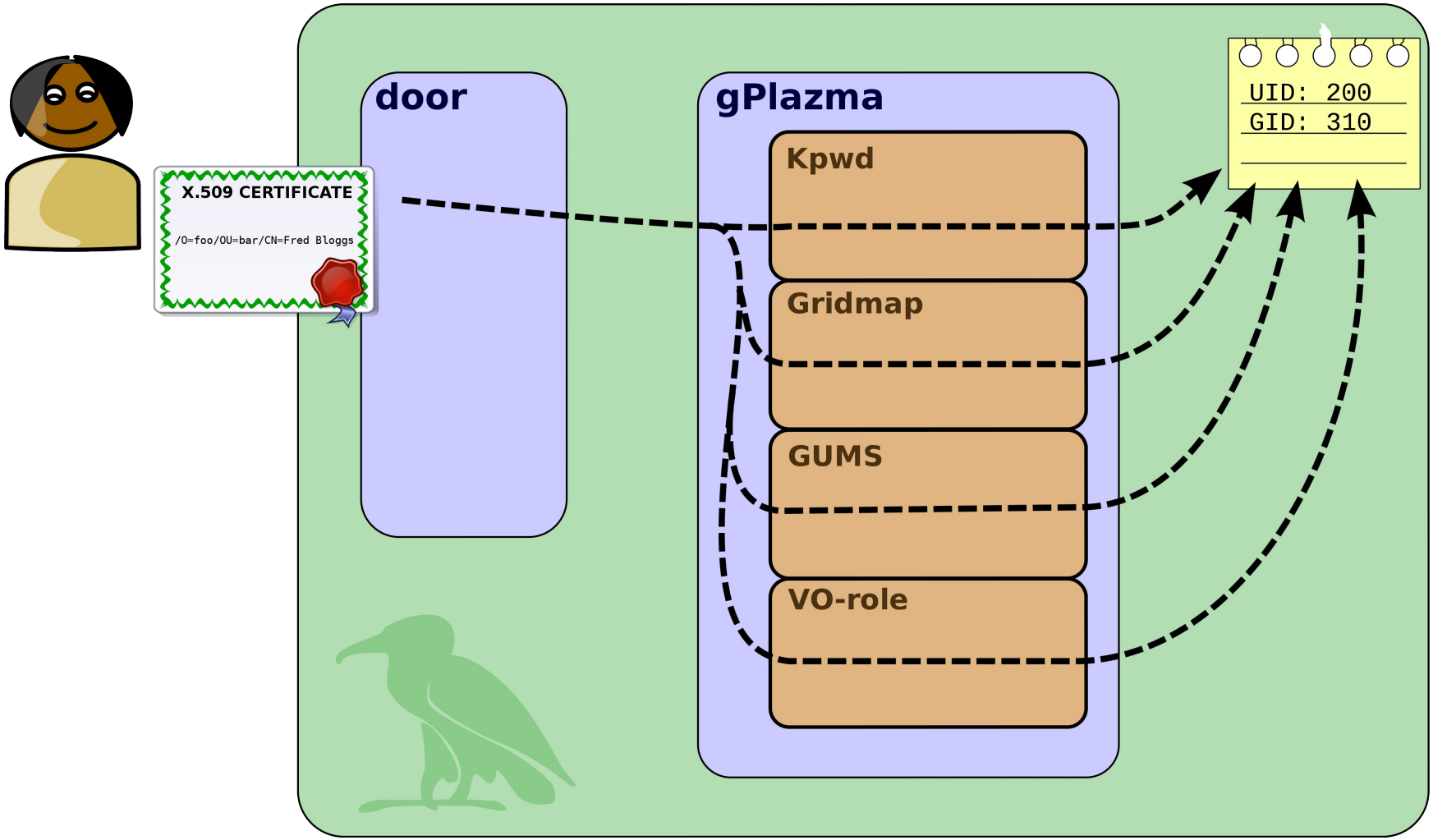
(shouldn't that be "gPlazmo" ???)

= the dCache native multi-protocol authorization framework.

Version 1 in process of being replaced by version 2.

We provided the XACML plugin (adapted from version 1).

gPlazma v1 structure



Authorization in gPlazma2: "Phases" vs "Plugins"

PHASES

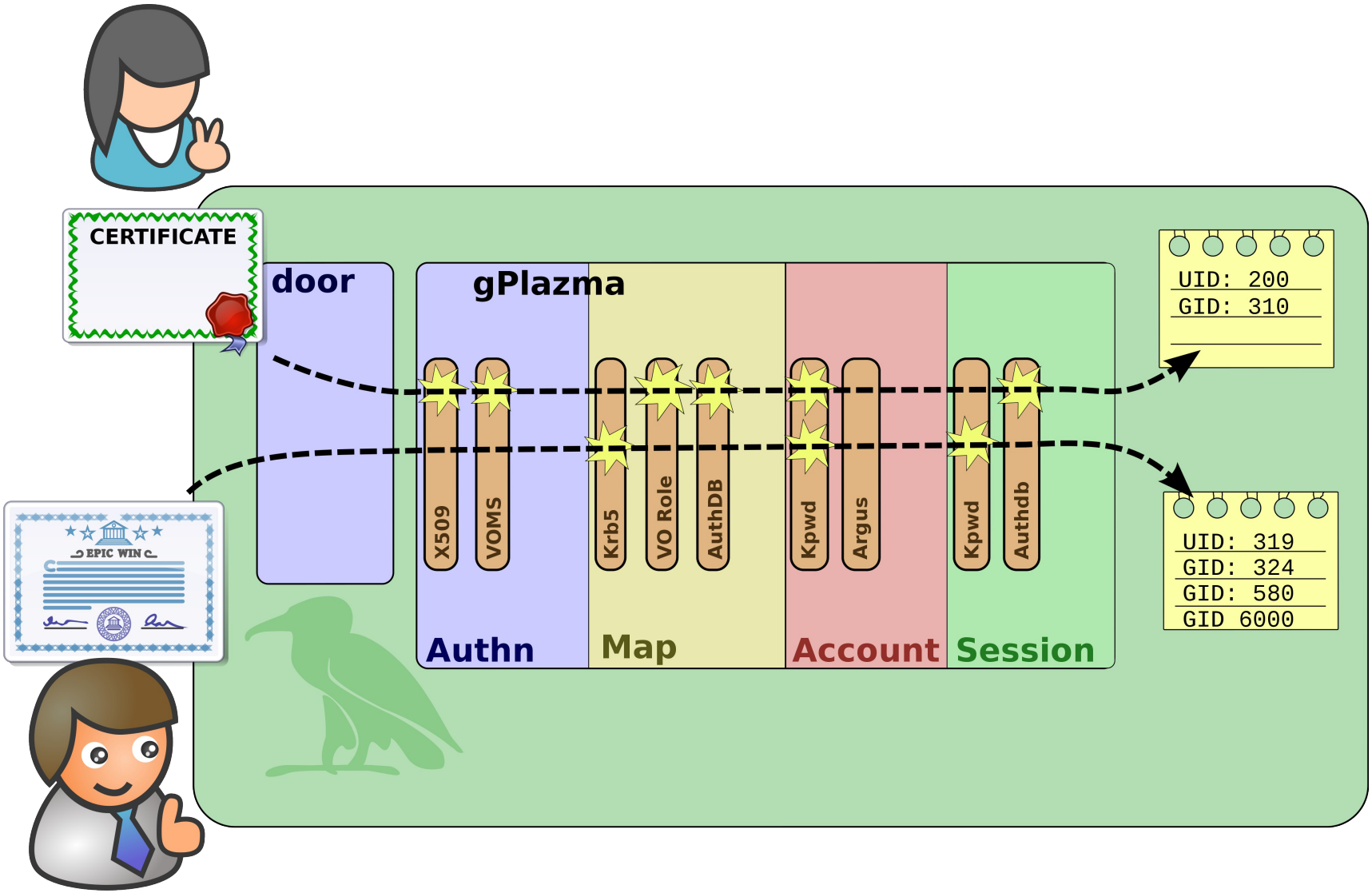
- *Authenticate* – "Who are we talking to?"
- *Map* – "How does the authenticated user fit into our site?"
- *Account* – "Is the account currently banned?"
- *Session* – "What is the user allowed to access?"

PLUGINS

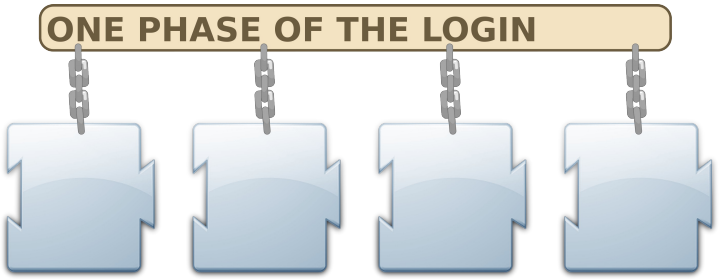
Protocol-based: X509, VOMS, XACML, KPWD, KRB5, VORole, AUTHZDB, etc.

Plugins can implement more than one phase.

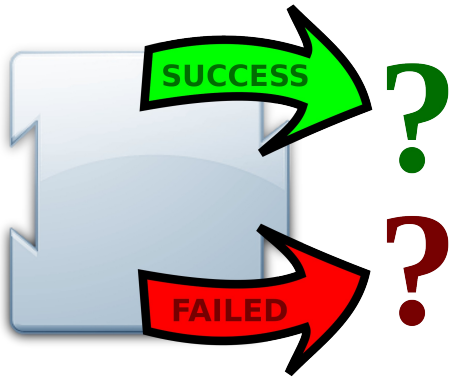
Logging in: plugins



Wiring plugins together



- Each phase is comprised of plugins
  - Result of a phase depends on the result of running these plugins
- When a plugin runs:
  - Running a plugin is either **success** or **failure**.
  - Plugins that fail sometimes is expected
- Four options describe what to do next:



Name	Description
`optional`	Success or failure of the plugin doesn't matter; always move onto next one in the phase
`sufficient`	Successful plugin finishes the phase with success
`requisite`	Failing plugin finishes the phase with failure
`required`	Failing plugin fails the phase but remaining plugins are still run

Example of */etc/dcache/gplazma.conf* for Public dCache

```
auth     sufficient xacml  "gplazma.vomsdir.ca=/etc/grid-security/certificates" \
                           "gplazma.vomsdir.dir=/etc/grid-security/vomsdir" \
                           "gplazma.xacml.service.url=https://gums.fnal.gov:8443/gums/services/\
                            GUMSXACMLAuthorizationServicePort"

map      sufficient authzdb
session  optional   authzdb

auth     optional x509

auth     sufficient kpwd  "kpwd=/etc/dcache/dcache.kpwd"
map      sufficient  kpwd  "kpwd=/etc/dcache/dcache.kpwd"
session  sufficient  kpwd  "kpwd=/etc/dcache/dcache.kpwd"
```

Plugins run in the order in which they are defined, per phase; hence: auth (xacml, x509, kpwd); map (authzdb, kpwd); session (authzdb, kpwd)

Further Details

- http://www.dcache.org/manuals/2012/workshop/slides/gPlazma_theory.pdf

- http://www.dcache.org/manuals/2012/workshop/slides/gPlazma2_configuration.pdf